基于历史模拟法的风险价值算法在GPU上的实现和优化

张 劼[1] 文敏华[2] 林新华[2] 孟德龙[2] 陆 豪[1]
(上海清算所 上海 200001)[1] (上海交通大学高性能计算中心 上海 200240)[2]

摘 要 风险价值(Value at Risk,VaR)是风险管理的基本工具,可对现有头寸的下行风险提供量化衡量方法。基于历史模拟法的VaR(Historical VaR)是最流行的计算方法之一,被广泛应用于世界各大金融机构。对金融产品进行实时或准实时的VaR计算,对于及时规避金融风险具有重要意义。由于金融产品日益复杂,产品数量持续增长,现有CPU计算平台上的计算能力已经难以满足VaR的性能需求。为解决这一问题,在GPU上使用CUDA对Historical VaR的计算代码进行了实现和优化。通过改进排序算法、基于Multi-stream隐藏通讯时间、解耦数据依赖并实现细粒度并行等优化方法,CUDA版本的VaR计算性能比优化后的CPU单核性能提升了42.6倍,为快速计算超大数量债券的VaR提供了有效的解决方案。以上优化方法也可以为金融领域内其他算法的GPU化提供思路。

关键词 风险价值,Historical VaR,CUDA,风险管理
中图法分类号 TP399 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2018.05.050

Implementation and Optimization of Historical VaR on GPU

ZHANG Jie[1] WEN Min-hua[2] Jame LIN[2] MENG De-long[2] LU Hao[1]
(Shanghai Clearing House,Shanghai 200001,China)[1]
(Center for HPC,Shanghai Jiao Tong University,Shanghai 200240,China)[2]

Abstract Value at Risk(VaR) is a fundamental tool for risk management,providing a quantitative measure of the downside risks to existing positions. Historical VaR is one of the most popular methods of VaR calculation,which is widely used in many financial institutions in the world. Real time or quasi-real-time VaR calculation for financial products is quite important to avoid financial risks in time. Due to the increasing complexity and increasing number of financial products,the computational capability of existing CPU platforms has not been able to meet the requirements of risk management for computing speed. To solve this problem,the historical VaR method was implemented and optimized on GPU by using CUDA. By improving the sorting algorithm,overlapping the communication time using Multi-stream,decoupling the data dependency and implementing the fine-grained parallel optimization method,42.6x speedup is achieved when it is compared with the performance of the single core CPU,which provides solution for fast VaR calculation of large amount of bonds. The optimization methods of this paper can also be referenced by other financial algorithms on GPU.

Keywords Value at risk,Historical VaR,CUDA,Risk management

1 引言

1993年,G30集团在研究衍生品种的基础上发表了题为《衍生产品的实践和规则》[1]的报告,提出风险价值方法。目前,VaR已经成为金融界测量风险的主流方法。VaR为现有头寸的下行风险提供量化衡量方法,直观地给出了在一定的置信度水平下与一定的目标水平之上预期的最大损失(或最坏的损失),即"有$X\%$的把握,在今后的N天内损失不会大于V"。其中,数量V就是投资组合的VaR。为保障风险测量的精确性,VaR需要计算不同价格水平下资产的潜在损失,这需要庞大的计算量。

近年来,GPU(Graphics Processing Unit)已被广泛运用于科学计算和工程计算领域的加速。相比于CPU,GPU拥有更强的计算能力和更大的数据读写带宽,二者在同时期的主流性能差别往往能达到一个量级。不仅如此,GPU与CPU的硬件计算能力还呈现不断扩大的趋势。CUDA(Com-

到稿日期:2017-02-22 返修日期:2017-04-14 本文受NVIDIA GLOBAL CENTER OF EXCELLENCE,NVIDIA GPU全球卓越中心项目资助。

张 劼(1988-),男,硕士,工程师,主要研究方向为金融计算、智能信息处理,E-mail:zhangjie@shclearing.com(通信作者);文敏华(1988-),男,硕士,工程师,主要研究方向为高性能计算,E-mail:wenminhua@sjtu.edu.cn;林新华(1979-),男,硕士,高级工程师,主要研究方向为高性能计算,E-mail:james@sjtu.edu.cn;孟德龙(1992-),男,硕士生,主要研究方向为高性能计算,E-mail:mdlong@sjtu.edu.cn;陆 豪(1989-),男,硕士,主要研究方向为债券估值,E-mail:luhao@shclearing.com。

pute Unified Device Architecture)的发布使得编程人员可以在不了解图形学背景的情况下对 GPU 进行高效率的编程,从而使 GPU 的软件生态环境有了质的提升;同时,基于制导语句的编程模型(如 OpenACC)的发展,也使得 GPU 编程得到了进一步简化。

GPU 的强大计算能力和良好的生态系统构建方式,已经使其成为加速各类应用程序的主要工具和解决金融风险管理瓶颈的一种重要研究方法。目前已经有大量基于 GPU 的金融计算解决方案[2-4]。本文对基于历史模拟法的 VaR 的计算代码进行了 GPU 实现和优化,并获得了显著的性能提升,CPU 单核性能提升了 42.6 倍,对金融风险的监控有很大的现实意义,同时对其他金融程序的 GPU 化也有一定的指导意义。

本文第 2 节介绍相关工作;第 3 节介绍基于历史模拟法的 VaR 计算方法的实现;第 4 节介绍该算法的具体 GPU 实现和优化;第 5 节为优化结果及其讨论;最后总结全文。

2 相关工作

目前已经有大量基于 GPU 的金融计算解决方案。Abbas-Turki 等[5]基于蒙特卡洛方法在 GPU 上进行了金融衍生品加速的研究,获得了 2～10 倍的性能提升。Fatica[6]在 GPU 上对基于最小二乘法的蒙特卡洛进行了加速,相比于 OpenMP 并行后的 CPU 程序,Fatica 的方法获得了 3.3～5.4 倍的加速。

通常,VaR 计算包含 3 种方法:历史模拟法、蒙特卡洛方法和压力测试法。目前 VaR 在 GPU 上的优化工作主要是基于蒙特卡洛方法[7-8],这主要是由于蒙特卡洛在计算未来损益分布时采用了随机过程模拟,在 GPU 上极易并行。Wood[9]在 GPU 上实现了基于历史模拟法的 VaR,但仅仅是在 GPU 上实现了金融产品级别的粗粒度并行,这种简易的并行方式只有在模拟的金融产品数量非常庞大时才具有较好的加速效果,这种并行策略限制了其在实际应用中的适用性,而且其工作平台基于第一代的 GPU 加速卡 Tesla C1060,优化策略与性能结果已无法作为参考。

本文的主要贡献如下:

1)基于生产环境代码,在 GPU 上实现并优化了基于历史模拟法的 VaR 程序,获得了 42.6 倍的性能提升;

2)考虑到生产环境中的输入数据需要从 MySQL Server 中读取,使用了多流的方式,让该 GPU 程序使用 Hyper-Q 特性,相比未使用该特性时性能有了显著提升;

3)将单个债券的 VaR 计算数据依赖进行了解耦,实现了持有期层面和回溯天层面的并行,增加了计算并行度,使得 GPU 的计算效率得到进一步提升。

据我们所知,目前尚无其他历史模拟法在 GPU 的工作上使用了 2)和 3)的优化方法。

3 基于历史模拟法的 VaR 模型

历史模拟法通过对各种情景下的投资组合进行重新定价来衡量风险,属于完全估值法(Full-valuation Method)。其核心在于根据风险因子的历史样本变化来模拟证券组合的未来损益分布,并利用分位数给出一定置信度下的 VaR 估计。基于历史模拟法的 VaR 的计算流程如图 1 所示。

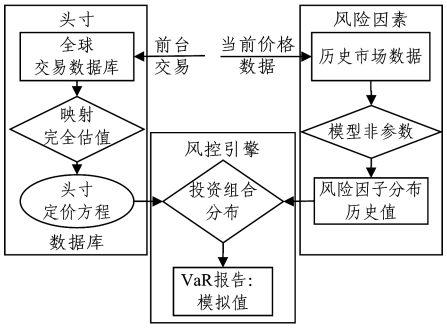

图 1 基于历史模拟法的 VaR

Fig. 1 Historical VaR

在模拟计算中,第 1 个抽样是假定市场变量的百分比变化等于数据库所覆盖的第 1 天数据的百分比变化;第 2 个抽样是假定市场变量的百分比变化等于数据库所覆盖的第 2 天数据的百分比变化;依此类推。

对于每个样本,可以计算出投资组合的变化值 ΔP,最终可以从 ΔP 的概率分布的分位数中求得投资组合的 VaR。

计算过程中,需要分别对风险敞口(PiN)和风险因子的分布(Vol)建模。

1)首先,需要将投资组合中的每个资产映射到简单和标准化的模型中。本文使用每月收益来计算两个连续时间段之间每个资产的回报,如图 2 所示。

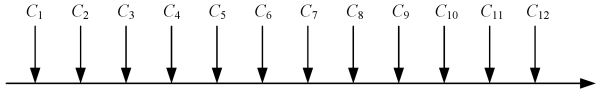

图 2 券现金流示意图

Fig. 2 Schematic diagram of bond cash flows

2)然后,将每个资产的历史回报加入其当前价值,并用资产的总价值来衡量整个投资组合。

$$PiN=\sum_{i=1}^{m}\frac{CF_i}{[1+r_i'+(r_{i+T}-r_i)+Sspot]^{t_i}}$$

其中,CF_i 是现金流,m 是现金流条数,r_i'为即期收益率,r_i 为历史收益率,$Sspot$ 为即期利差。

3)接着,计算每个回溯日的损失。由于 VaR 是正常市场条件和置信区间 $X\%$下的最大损失,因此需要将损失 Vol 由高到低排序,VaR 关注的是排序后的数组首段。

$$Vol_i=\max(0,P-P_{i,N_{\max}},P-P_{i,N_{\max-1}},\cdots,P-P_{i,1})$$

4)最后,根据置信区间在排序好的 Vol 数组中确定最终的 VaR 值。

$$VaR=Vol_{(1-\alpha)\times Backdays}$$

由于 VaR 关注的是排序后的数组首段,而对所有的 Vol 排序是一种比较低效的方式,因此我们在标准历史模拟法 VaR 的基础上对排序方法进行了优化。根据置信区间和回溯期长度维护一个大小为 k 的最小堆,来计算最终的 VaR。当持有期增加时,只需将新的 Vol 与堆的顶点进行比较即可,如果新的 Vol 较大,则将其添加进最小堆,并删除顶点,如图 3 所示。时间复杂度由 $O(n*\log n)$ 降为 $O(n*\log k)$,排序效率得到极大改善,而且该优化方法提升了 CPU 和 GPU 平台的性能。

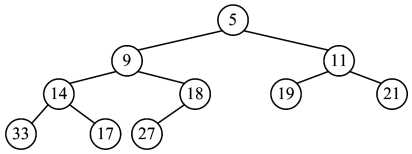

图 3　VaR 排序堆示意图

Fig. 3　Heap sort in VaR

4　GPU 程序的实现和优化

4.1　VaR 算法的并行可能性分析

近年来,计算机计算能力的不断提升主要得益于多核和众核架构的发展。为了进一步提升 VaR 模型程序的计算性能,需将 VaR 算法并行化,以利用众核计算机的计算能力。

首先,对基于历史模拟法的 VaR 算法进行并行可能性分析,下面给出 3 种不同粒度的并行方案。1)在债券层面并行时,算法中涉及到多个债券的计算,而各个债券的计算没有相关性,因此该算法在债券层面是可并行的。2)在持有期层面并行时,每个债券的计算中涉及到多个持有期的计算,而每个持有期依赖之前持有期的计算结果,因此可以先将各个持有期进行并行计算得出结果,再经过一次数据同步来保证结果的正确性。此并行方案会对每个债券的计算产生一次线程同步的额外开销。3)在回溯天层面并行时,每个持有期的计算包含对多个回溯天的风险敞口的计算,此层面的并行将不能使用前文提出的线性堆排序算法,根据置信区间确定风险价值的过程将由并行全排序来完成。

在 3 种并行方案中,在回溯天层面并行的并行度最高,但会产生最多的额外开销;在债券层面并行的并行度最低,但此方案不会产生额外开销;而在持有期层面并行可以获得较高的并行性,同时只产生较少的额外开销。不同的方案具有不同的特性,可以根据不同的生产环境及计算需求来选取不同的并行方案。例如,在离线计算的生产环境中,影响 VaR 计算系统性能的关键因素为计算吞吐量,此时有足够数量的债券来保证足够的并行度,因此可以选取额外开销最少的第一种并行方案。而在高频的线上实时计算环境中,影响系统性能最关键的因素为响应速度,此时单次计算任务的债券数量较少,可以选取并行度最高的第三种并行方案,虽然产生了部分额外开销,但此方案的高并行性能更充分地利用了多核计算机的计算能力,能最快完成计算任务。

其次,分别分析 VaR 模型在 CPU 和 GPU 平台上的不同计算特性。GPU 平台的核心数量较多,单核计算能力较弱;CPU 平台的核心数量较少,单核计算能力较强。根据上文所述,VaR 模型有多种粒度并行方案,对于 CPU 计算平台而言,常见的单台服务器能提供的多核数量为 32 左右,方案 1)的并行粒度已能在大多数场景中利用其计算能力,且 CPU 平台的性能主要依赖于单核性能,同步操作会大大降低 CPU 平台的性能;对于 GPU 平台而言,常见的单卡核心数量在 1000 以上,方案 2)和方案 3)仅额外需要部分核心在债券层面同步,因此可以减少同步操作带来的额外开销。由上述可知,各并行方案均能在 GPU 平台上实现。此外,单台服务器可以配备 4 块甚至更多的 GPU 计算卡,可以使用 GPU 计算卡进行计算,使用 CPU 处理上层系统的数据收发及整合调度任务。因此,基于 GPU 平台的计算服务器能提供更合适的资源配置及密度更高的计算平台。

4.2　VaR 算法的 GPU 实现与优化

根据 4.1 节的算法可并行性分析,对 VaR 算法进行不同层次的实现和优化后作为最初的 GPU 实现,并在债券层面进行并行移植,这也是最容易实现的 GPU 方案;在此基础上,进行细粒度并行优化,包括持有期层面并行和回溯天层面并行。

4.2.1　*债券层面并行*

在债券层面的并行方案中,每个线程负责计算一个债券的风险价值。GPU 中有上千个线程同时执行,因此多个债券一起加载到 GPU 上计算可以获得较高的效率。首先在 CPU 端进行数据初始化,读取所有需要计算的数据,然后将这些数据统一传输到 GPU 上进行计算。该方法只有在所计算的通信比很大时才有较好的性能,否则会由于数据传输而带来较大的额外开销。另外,在实现中,为了与生产环境相匹配,GPU 上使用了与 CPU 一致的数据结构,这会造成访问不连续,从而影响性能。

4.2.2　*持有期层面并行*

在债券层并行方案中,因为每个线程负责一个债券的计算,当计算任务比较少时,GPU 的计算资源并不能被完全利用,所以相对于 CPU 程序,GPU 程序的加速并不理想。为了应对少量任务的场景,我们实现了细粒度的并行优化方案以提高程序的并行度。

对于原本一个线程的计算量需要分配给多个线程计算的问题,对原有任务进行拆分,以实现更细粒度的并行。各个债券的计算过程会涉及多个现金流以及每个现金流的历史数据的计算,然后统计分析风险价值。这里可以由一个线程计算一个完整的持有期,因此特定持有期内的风险价值只有一个线程负责计算,不会存在多线程数据读写冲突问题。置信区间通常高于 90%,即只需要存储历史风险价值较高的前 10% 的数据,因此在每个持有期只有 10% 的数据会被最终存储到二叉堆中。尽管每个债券仅有 10 个持有期,但相比于债券层并行方案,其并行度已大大提高。该方案中,每个 Block 同时处理 25 个债券,每 10 个线程处理一个债券。

4.2.3　*回溯期层面并行*

在持有期层面并行方案中,每个债券的计算由 10 个线程并行计算,目前 GPU 的基本执行单位为每 warp 有 32 线程同时并行执行,这势必使得不同的债券被分配到同一个 warp 执行并形成分支,降低了 GPU 的运行效率。考虑到每个债券依据 250 个回溯天的数据进行风险因子的计算,我们实现了回溯天层面的并行,即在每一个线程块中使用 250 个线程进行一个债券的计算。

由于每个回溯期的风险因子的计算相互独立,因此这部分可以很方便地使用多线程并行。计算完风险因子后,需要选取最大的 25 个风险因子并排序,这里使用 GPU 并行版本的双调排序方法对 250 个回溯期的风险因子进行完全排序,具体实现可以参照 CUDA Sample 里的双调排序算法。为了提升数据的读写性能,风险敞口数据存储于 GPU 上的高速存储 shared memory。

4.3　Multi-stream 优化

CUDA 流表示一个 GPU 操作队列,并且该队列中的操

作依其被添加到队列的先后顺序执行。使用 CUDA 流可以实现任务级的并行，例如 GPU 在执行核函数的同时，还可以在主机和设备之间交换数据。可以利用 CUDA 的此特性，将任务分批次传输到 GPU 进行计算。Kepler 架构新引入的 Hyper-Q 技术也使得多个 GPU 核函数同时计算时可以获得较高的效率。

在回溯期层面并行的基础上，实现了 Multi-stream 优化。以 16000 个债券为例，可以将其分为多块，如每块 1000 个债券，用一个 CUDA 流计算每个块，这样即分成了 16 个 CUDA 流。创建 CUDA 流后，CPU 每次传输 1000 个债券的数据到 GPU，然后进行计算，后一个 CUDA 流不需要等待之前的 CUDA 流完成计算即可以开始数据传输，通过这种方式可以将不同的 CUDA 流之间的数据传输和计算重叠，从而隐藏数据传输时间并提高性能。具体方式如图 4 所示。

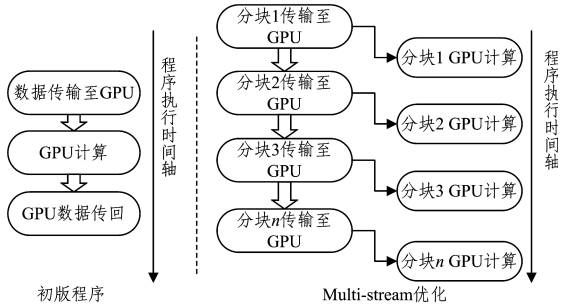

图 4 Multi-stream 优化示意图

Fig. 4 Multi-stream optimization

5 结果分析

实验环境如表 1 所列，使用 Intel Haswell E5-2693V3 来搭载 NVIDIA Tesla K80 GPU。本文使用的基准 CPU 程序基于标准历史模拟法，排序则基于优化后的排序算法，使用 Intel 编译器并使用 Ofast 选项进行编译。

表 1 程序运行环境

Table 1 Running environment of program

项目	配置
操作系统	CentOS 6.6 x86_64
CPU	14-core 2.30 GHz Intel Haswell E5-2693 V3
CUDA	NVIDIA CUDA-7.5
GPU	NVIDIA Tesla K80 GPU
编译环境	ICC/NVCC
编译选项	-Ofast
操作系统	CentOS 6.6 x86_64

本节使用两个不同的算例来评估程序的性能。算例配置如表 2 所列。

表 2 算例配置

Table 2 Tests configurations

算例	总债券数量	每块债券数量	每债券持有期数
A	16000	1000	10
B	1000	250	10

其中，算例 *A* 的债券数量较多，用于评估计算大量债券时程序的性能。算例 *B* 的总债券数量较小，可用于评估计算量较小时的计算响应时间。

采用前文所述的优化方法后，算例 *A* 及算例 *B* 的性能分别如图 5 和图 6 所示。需要说明的是，出于对数据安全的考虑，本文在测试中并没有使用市场的真实数据，而是通过随机数构造接近于市场真实环境的模拟数据，因此结果中数据的初始化过程占据了大量时间；但在生产环境中，数据在内存中产生，这部分时间开销将大大减少。本文使用程序中数据传输和计算的时间总和来评估程序的性能（即总时间除去初始化时间）。

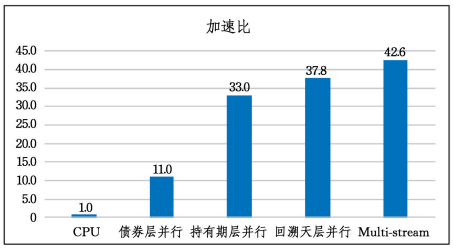

图 5 算例 *A* 的加速比

Fig. 5 Speedup of test *A*

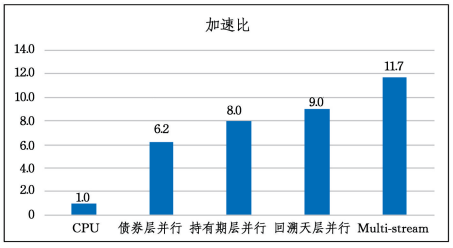

图 6 算例 *B* 的加速比

Fig. 6 Speedup of test *B*

对于算例 *A* 而言，债券层面的 GPU 并行仅获得了 11.0 倍的加速比；对于算例 *B* 而言，GPU 化所取得的加速比更低，其原因是算例 *B* 的总债券数量较少，而该版本的 GPU 程序使用一个线程来计算一个债券，少量的债券数量不能充分利用 GPU 的计算能力。

经过更细粒度的并行优化，即持有期层面和回溯期层面并行，程序的性能得到了进一步提升。对于算例 *A* 和算例 *B*，并行度的提高都能带来性能的提升，对 GPU 的硬件利用效率也更高。

采用 Multi-stream 优化之后，两个算例较初始 GPU 版本都有一定的性能提升。

最后，通过一系列的优化，对于规模更大的算例 *A*，其性能提升了 42.6 倍。

结束语 本文将基于历史模拟法的 VaR 方法在 GPU 上进行了移植和优化。VaR 计算单个债券的计算量大，计算包含了复杂的排序过程，限制了程序的并行度。我们在 GPU 程序中使一个债券的 VaR 计算在 GPU 的 block 内并行实现，极大地提高了程序的并行度，使得程序的性能提升了42.6倍。同时，考虑到在生产环境中 VaR 的初始数据是在 MySQL 中持续不断生成的，我们使用了 Multi-stream 的方法，并利用了 Hyper-Q 的特性，使得数据传输的时间可以被计算时间隐藏，进一步提升了程序的性能。

VaR 的计算在 GPU 上的运行速度获得了 1 个量级以上

（下转第 321 页）